Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 5546581, 15 pages
https://doi.org/10.1155/2021/5546581

Research Article

Autonomous Last-Mile Delivery Based on the Cooperation of Multiple Heterogeneous Unmanned Ground Vehicles

Yuzhan Wu,[1] Yuanhao Ding,[2] Susheng Ding,[2] Yvon Savaria,[3] and Meng Li[4]

[1]*Beihang University, Beijing, China*
[2]*School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China*
[3]*Department of Electrical Engineering, Polytechnique Montreal, Montreal, QC, Canada*
[4]*College of Big Data and Internet, Shenzhen Technology University, Shenzhen, China*

Correspondence should be addressed to Meng Li; limeng2@sztu.edu.cn

Received 29 January 2021; Revised 17 February 2021; Accepted 28 February 2021; Published 12 March 2021

Academic Editor: Xiaobo Qu

With the development of e-commerce, the last-mile delivery has become a significant part of customers' shopping experience. In this paper, an autonomous last-mile delivery method using multiple unmanned ground vehicles is investigated. Being a smart logistics service, it provides a promising solution to reduce the delivery cost, improve efficiency, and avoid the spread of airborne diseases, such as SARS and COVID-19. By using a cooperation strategy with multiple heterogeneous robots, contactless parcel delivery can be carried out within apartment complexes efficiently. In this paper, the last-mile delivery with heterogeneous UGVs is formulated as an optimization problem aimed at minimizing the maximum makespan to complete all tasks. Then, a heuristic algorithm combining the Floyd's algorithm and PSO algorithm is proposed for task assignment and path planning. This algorithm is further realized in a distributed scheme, with all robots in a swarm working together to obtain the best task schedule. A good solution with an optimized makespan is achieved by considering the constraints of various robots in terms of speed and payload. Simulations and experiments are carried out and the obtained results confirm the validity and applicability of the developed approaches.

1. Introduction

Unmanned ground vehicles (UGVs) have been successfully applied in diverse applications, such as planet exploration on the Moon and Mars [1, 2]. In recent years, UGVs have also been used for logistic services to reduce delivery costs [3]. Benefiting from autonomous technologies, UGVs can operate 24 hours per day and 7 days per week, which provides flexibility in terms of service time compared with conventional employee scheduling. Given the environment in which UGVs run, task assignment and path planning should be performed considering the parameters of parcels and UGVs. However, it is challenging to find an optimal solution with multiple UGVs under physical constraints. This problem is known as the vehicle routing problem (VRP), which was proved to be NP-hard [4].

The VRP model was first proposed by Dantzig and Ramser in 1959 [5] and has received increasing attention since then. A typical VRP aims at optimizing product delivery by vehicles from one depot or multiple depots to customers under certain constraints. There exist many extensions of the VRP, which can be classified into different categories according to configurations, problem modelling, solving algorithms, and objectives [6–16]. For example, this problem can be formulated differently considering homogeneous vehicles or heterogeneous vehicles [6, 7]. It can be further extended by adding different constraints to the original definition such as time constraints, traveling distance constraints, and capacity constraints [11, 12, 15, 17].

Typically, at the algorithm level, a VRP to solve is abstracted as a graph associated with vertices and arcs, such as a directed graph, an undirected graph, a connected graph, or a weighted graph. In the last decades, many algorithms were proposed and investigated, which include those based on branch and bound, cutting planes, dynamic programming,

and heuristic approaches. Due to the complexity of VRPs, heuristic algorithms often perform better in terms of computing time and flexibility and therefore are often preferred in practical applications. In [14], genetic algorithms are applied to solve the multidepot VRP, considering total traveling distance and total traveling time as two objectives. In [4], enhanced ant colony optimization (ACO) algorithms are proposed, which allow ants visit the depots more than once until they reach all customers. In [18], a heuristic algorithm combining a variable neighborhood search algorithm and a space saving heuristic algorithm is proposed for logistic optimization. However, most researches mainly considered homogeneous UGVs which are not always true in reality.

In this paper, the multiple heterogeneous UGV model is adopted for last-mile delivery within apartment complexes. The delivery service is divided into the VRP and the bin packing problem to achieve better performance. Utilizing the distributed feature of the PSO algorithm, we have developed a coordinate controller that can be installed within UGVs. The design of the controller is shown in Figure 1. Each controller consists of two parts, namely, the multinode coordinator and the PSO optimizer engine. This engine can be treated as a regular PSO optimizer that iterates to search in the solution space to obtain the best solution. The multinode coordinator acts as a lightweight coordinator for the whole team. Each controller has a multinode coordinator, while at any given time, only one coordinator is active, so that the whole team follows the guidance of one coordinator.

The coordinator takes care of the following tasks:

(i) Generating a leader coordinator by competition;

(ii) Sending messages to start and stop the optimization process;

(iii) Unifying the best solution that is assigned to each UGV.

The main contributions of this paper can be summarized as follows:

(i) Formulate the last-mile delivery with heterogeneous UGVs as an optimization problem aimed at minimizing the maximum makespan;

(ii) Propose an algorithm combining the Floyd's algorithm and PSO algorithm for task assignment and path planning;

(iii) Propose a distributed logistic control algorithm validated using semiphysical simulations and experiments.

The remainder of this paper is organized as follows. Some existing techniques are reviewed in Section 2. Section 3 presents the proposed mathematical formulation of the last-mile delivery problem with multiple heterogeneous UGVs. The proposed algorithm is introduced in Section 4. Coordinate controller for distributed logistic with multiple UGVs is given in Section 5. Validation experiments are reported in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

Planning the shortest path is a key issue for vehicles in a transport system. Various methods have been proposed so far, such as graph search-based planners, sampling-based planners, interpolating curve planners, and numerical optimization approaches [19]. In the following subsections, we mainly focus on graph search-based shortest path algorithms, heuristic searching algorithms, and bio-inspired optimization algorithms.

2.1. Graph Search-Based Shortest Path Algorithms. There exist many algorithms to solve the shortest path problem in a graph, among which we can find the Dijkstra's algorithm, the Bellman–Ford algorithm, and the Floyd's algorithm.

The Dijkstra's algorithm was proposed by Eds Wybe Dijkstra in 1959 [20]. It uses a greedy algorithm model and is a typical alternative to the shortest path method. It is one of the most commonly used algorithms to find the shortest path in graph theory. The main idea of this algorithm is to calculate all paths from a certain point to other vertices, select the shortest path, and then add the point to the final shortest path.

The main difference between Dijkstra's algorithm and Bellman–Ford algorithm is that the latter can use negative weights to determine the distance value [21]. The Bellman–Ford algorithm is mainly used to solve the shortest path from a single source and the distance between two points in a directed graph. If there is a loop between these two points, in order to prevent the shortest path to be calculated continuously, the Bellman–Ford algorithm can identify negative cycles and deal with the negative loop problem. Because the Bellman–Ford algorithm restarts from the same source point to perform the "relaxation" update operation every time, it is slower than Dijkstra's algorithm for the same problem.

Floyd's algorithm is a classic dynamic programming algorithm and was first published in 1962 [22]. It is an algorithm to solve the shortest path between any two points in a weighted graph. It can correctly handle the shortest path problem of directed graphs comprising some negative weights, and it is also used to calculate the transitive closure of directed graphs. In [23], the author improved the Floyd's algorithm by replacing fixed weights with probability time-consuming weights. The least time path is calculated based on the edge probability and the time weight, which improved the speed of the Floyd algorithm.

To summarize, the Dijkstra's algorithm adopts a greedy approach to calculate the shortest path, whereas both the Bellman–Ford algorithm and the Floyd's algorithm use dynamic programming.

The algorithms mentioned above have been widely used in seeking the shortest path for vehicles in transport systems, with modifications made according to the application background. In [24], a modified Dijkstra's algorithm has been proposed for route planning in public transport systems. The algorithm takes the number of transfers and displacement distances into account, making it superior to

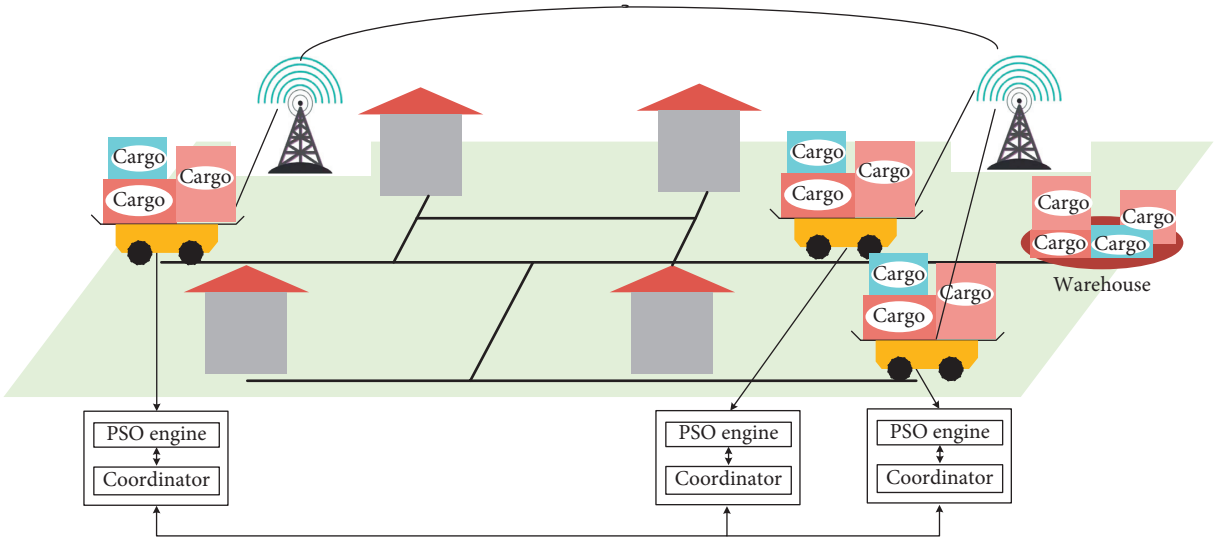

Figure 1: Design of coordinate controller.

the classic Dijkstra's algorithm. In [25], the Dijkstra's algorithm is modified to find the maximum load path to improve the freight transport efficiency. In [26], the Dijkstra's algorithm is combined with the impedance function model to find the shortest path in a parking lot, taking the dynamic time of routes into account. These methods are further combined with heuristic searching algorithms. In [27], the Floyd's algorithm is combined with the genetic algorithm (GA) for route planning in multimodal transport. The Floyd's algorithm is used to find the shortest path and the GA algorithm is used to obtain the optimal transport mode. Following a similar idea, in [28], the Dijkstra's algorithm is combined with an ant colony optimization to search the path for robots. The Dijkstra's algorithm is applied to initial path planning, while a modified ant colony optimization algorithm is then applied to optimize the initial path. The algorithm proposed in our paper is also a combination of a classic shortest path algorithm enhanced with heuristic searching algorithm, combined with constraints of application background taking into account.

2.2. Heuristic Searching Algorithms. Theoretically, a graph search-based algorithm can usually provide the shortest path. However, with the increase of the scale of the problem, the computing time needed becomes intolerable. Heuristic searching algorithms are then developed to provide a solution within acceptable computing time. This is achieved by guiding the search direction with some kind of heuristic information. From this perspective, heuristic searching methods can be treated as an extension of graph search planners. The heuristic searching algorithms are usually used to solve practical applications. For example, a heuristic method was proposed to improve efficiency and merging capacity for highway T-junctions in [29].

Some significant representative heuristic searching algorithms are the $A*$ algorithm and its dynamic version, the $D*$ algorithm [30]. Extension of these algorithms includes the field $D*$ [31], the anytime repairing $A*$, and the anytime $D*$ [32]. So far, these methods have been widely used for path planning of different kinds of delivery vehicles. $A*$ algorithm is used to guide UAVs to find a safe path in battle fields [33], and $A*$ algorithm is combined with artificial bee colony to guide the motion of multiple UAVs in a 3D environment [34]. In an underwater environment, the $A*$ algorithm is adopted to guide underwater robots to track chemical plume [35]. Heuristic searching algorithms also help UGVs achieve excellent performance in the DARPA Urban Challenge, where hybrid $A*$ [36] and $A*$ [37] were used in Stanford University's UGV and KIT's UGV, while $AD*$ was used in Boss, the winning vehicle [38]. In [39], a rule-based $A*$ algorithm is presented to find the shortest path in the graphical domain for the emergency vehicle lane preclearing problem. In [40], the deterministic $A*$ algorithm with a stepwise strategy is applied to solve a cooperative sorting problem for connected and automated vehicles in a multilane platoon.

2.3. Bio-Inspired Optimization Algorithms. Due to limitations of traditional path planning approaches, algorithms inspired by the observation of natural systems are developed to solve this NP complex problem [41], which attracts a growing interest of researchers. Among bio-inspired algorithms, the more widely used ones include the GA, the particle swarm optimization (PSO) algorithm, and the ant colony optimization (ACO) algorithm.

GA is a popular search-based algorithm, which was initially proposed by Bremermann in 1958 [42]. GA is a means to solve optimization problems. It generally relies on an objective function that must be maximized or minimized under specified constraints. In this case, a population of candidate solutions is assigned to a given problem, and individuals in the population are assigned adaptive capacity values according to the objective function. Individuals in the population are selected according to the environment, and gene transfer can be carried out by exchanging genetic

materials that define possible solutions. Mutations in the population ensure diversity and avoid premature convergence of the algorithm, which may lead to missing the optimal result. Compared with local algorithms, genetic algorithms can make better use of historical information. Shibata et al. proposed a strategy using GA in a static environment for the application of path planning [43]. In an unknown environment, GAs only need very little prior information to complete path planning and an implementation can be found in [44].

PSO is a heuristic algorithm based on natural phenomena, which was initially intended for simulating the social behavior of fish or birds. The basic idea was first proposed by Eberhart and Kennedy in 1995 [45] and soon became a popular optimization algorithm used to solve various problems in engineering and science. It does not require any individual leader and the required solution is obtained via guiding the particles' search by particles close to the food. Each particle considered through PSO represents a possible solution in the algorithm. At present, PSO is widely used in many fields, including multirobot positioning and path planning [46–49]. The use of PSO helps to reduce the computational effort to find a solution, maintain stable convergence characteristics, avoid falling into local optimum, and obtain higher-quality path planning results.

Ant system is an intelligent bionic optimization algorithm inspired by the foraging behavior characteristics of ant populations in nature by Italian researcher Dorigo [50]. It is a basic ACO algorithm, which provides a framework for other ACO algorithms. ACO has some drawbacks such as slow convergence speed, a tendency to fall into local optimum, and premature convergence. In response to these problems, many researchers have proposed effective improvements to the algorithm framework and structure, such as ant colony system [51, 52], max-min ant system [53], ant system with elitist strategy, and ant system with elitist strategy and ranking [54]. These improved algorithms have effectively improved the optimization ability, but a fixed pattern is used to update the pheromone and probability transfer rules, which lacks flexibility and fails to solve the problem of premature convergence.

In this paper, we combine the Floyd's algorithm with PSO algorithm to solve the last-mile delivery with multiple heterogeneous UGVs. The PSO algorithm is adopted to obtain the best task assignment solution for each UGV. The Floyd's algorithm is then applied to provide the waypoint sequence a UGV should follow. More details are presented in the following section.

3. Problem Formulation considering Multiple Heterogeneous UGVs

The problem to be solved in this paper can be modeled as follows. We use K robots to deliver G packages to L destinations from a warehouse. Robots are identified by an index k, ranging from 1 to K. Generally, the number of packages are not the same as the number of destinations, because there is a chance that some destinations receive no package, while some receive more than one package. As a result, G is generally different from L. In practice, we can ignore the destinations that receive no package. Meanwhile, if multiple packages must be delivered to the same destination, they can be treated as one big package. Thus, the problem of package delivery can be simplified and we get $L = G$. The maximum weight a robot can carry is b_k, $k = 1, 2, \ldots, K$, and the weight of each package to its destination is $w_i, i = 1, 2, \ldots, L$. The cost for robot k moving from the warehouse (represented by 0) or destination i to destination j is represented as $c_{ij}^k, i = 0, 1, \ldots, L, j = 1, \ldots, L$. The cost depends on the length of the path it chooses. The problem that needs to be solved is thus to spend minimum cost to deliver all the packages to their destinations. We define two more variables as follows:

$$
\begin{aligned}
y_i^k &= \begin{cases} 1, & \text{package } i \text{ is delivered by robot } k, \\ 0, & \text{otherwise}, \end{cases} \\
r_0^k &= \begin{cases} 1, & \text{robot } k \text{ needs to return to warehouse}, \\ 0, & \text{otherwise}, \end{cases} \\
x_{ij}^k &= \begin{cases} 1, & \text{robot } k \text{ moving to destination } j \text{ from } i, \\ 0, & \text{otherwise}. \end{cases}
\end{aligned} \tag{1}
$$

Thus, the problem can be modeled as minimizing total delivery cost. Then, we have

$$
\min C\left(x_{ij}^k, y_i^k\right) = \sum_{k=1}^{K} \sum_{i=0}^{L} \sum_{j=0}^{L} c_{ij}^k x_{ij}^k + \sum_{k=1}^{K} \sum_{i=0}^{L} r_0^k t_{i0}^k, \tag{2a}
$$

$$
\text{s.t.:} \quad \sum w_i y_i^k \le b_k, \quad \forall k, \tag{2b}
$$

$$
\sum_{k=1}^{K} \sum_{j=1}^{L} y_j^k = L = G, \tag{2c}
$$

$$
\sum_{i=1}^{L} x_{ij}^k = y_j^k, \quad \forall j, k, \tag{2d}
$$

$$
\sum_{j=1}^{L} x_{ij}^k = y_i^k, \quad \forall i, k, \tag{2e}
$$

$$
\sum_{i,j \in S \times S} x_{ij}^k \le |S| - 1, \quad S \in \{1, 2, \ldots, L\} \forall k, \tag{2f}
$$

$$
x_{ij}^k \in \{0, 1\}, \tag{2g}
$$

$$
y_i^k \in \{0, 1\}, \tag{2h}
$$

where t_{i0}^k stands for the shortest time for robot k returning to the warehouse from destination i. Equation (2b) implies that at one time, the weight of all packages carried by a robot should not exceed its ability. However, the weight of all packages allocated to a robot can exceed the ability of the robot in which case the packages are delivered through

multiple deliveries. Equation (2c) implies that all packages are delivered.

The solution of the abovementioned problem is a set of x_{ij}^k and y_i^k that can be encoded as

$$P_k = \left[p_1^k, p_2^k, \ldots, p_i^k, \ldots, p_L^k\right], \quad k = \{1, 2, \ldots, K\}, p_i \in [0, K), \tag{3}$$

where p_i^k stands for the case when the k th robot delivers a package to destination i. Based on this solution, we can further obtain the sequence of points that guides the movement of each robot.

In this paper, we are seeking to provide a schedule for all robots so that the task can be accomplished as fast as possible. The schedule must satisfy the following constraints:

(i) All packages are initially stored at a warehouse. Initially, the robots can be randomly deployed. After accomplishing the tasks, all robots need to return back to the warehouse.

(ii) The packages carried by a robot should not be heavier than the maximum payload of the robot.

In order to solve this problem, we combine the Floyd's algorithm with the PSO algorithm to find best solutions. The PSO algorithm is adopted to obtain the best task assignment. It specifies both the collection of packages delivered by certain robots and the order of delivering the packages. The Floyd algorithm is then adopted to provide the waypoint sequence the robot should follow.

To apply the PSO algorithm to solve this problem, the task assignment should be encoded as particles. Each particle is defined as a vector with multiple elements. The indexes of elements in one vector correspond to packages. The elements are generated randomly within a predefined range. The integer part of an element indicates the ID of a robot, and the fractional part indicates the order. The smaller the fractional part is, the earlier a package is delivered.

For example, if 2 robots are sent out to deliver 5 packages, a particle can be $[1.2, 2.5, 1.1, 2.3, 2.7]$.

This particle means that robot 1 delivers package 1 and package 3 since the first and third elements are 1.2 and 1.1, whose integer parts are 1. Package 3 should be delivered before package 1 because 1.1 is smaller than 1.2. Thus, the IDs of packages delivered by robot 1 are elements $[3, 1]$, and they are delivered following this order. Similarly, robot 2 is in charge of delivering elements $[4, 2, 5]$ with the corresponding order.

The fitness function can be given by

$$\min\left(\max\left(t_1, t_2, \ldots, t_N\right)\right), \tag{4}$$

where $t_i\,(i = 1, \ldots, N)$ is the time needed for robot i to finish its task. With this definition, the PSO algorithm can be applied to get the best task assignment solution. We then need to plan path for each robot. This is achieved by connecting the destinations of the packages. We use Floyd's algorithm to get the shortest paths between pairs of the destinations, and the path for robots can be obtained by connecting these paths.

4. Hybrid Algorithm for Task Assignment and Path Planning

This section introduces the algorithm to assign tasks and plan path for robots. The input of the algorithm is three matrixes: (1) the adjacency matrix of the map A^{map}, (2) the features matrix of the robots A^{robot}, and (3) the weight matrix of the packages A^{package}. The map is structured as a graph, with the warehouse and destinations being the nodes of the graph and paths connect nodes with edges. The weights of the edges are the length of corresponding paths. The feature matrix indicates the features of the robots. Each line indicates a robot, with the first row being speed, the second row being maximum payload, the third row being the time for the robot to pick up a package, and the fourth row being the time to drop a package. The weight matrix indicates the weight of the packages.

The output of the algorithm is N matrixes as A_i^{out}, $(i = 1, 2, \ldots, N)$, each corresponding to one robot. Each output matrix contains two rows. The first row contains a sequence of points of the way for the robot to follow, and the second row indicates the action the robot should perform at a specified waypoint.

The method proposed in this paper comprises three phases, as specified in Algorithm 1.

The first phase executes the classical Floyd's algorithm, as in lines 1 – 12. Taken A^{map} as input, it creates two matrixes as A^{distance} and A^{via}. Element $a_{i,j}^{\text{distance}}$ in A_{distance} is the shortest distance between node i and j, and element $a_{i,j}^{\text{via}}$ indicates the shortest path from node i to node j is via $a_{i,j}^{\text{via}}$.

The task sequence for each robot can then be obtained with the PSO algorithm, listed in lines 13–28. With the encoding scheme introduced in Section 3, the basic part of the PSO algorithm used here is the same as the classical one. The main difference is in the calculation of the fitness function. To calculate the fitness function of a particle, we first decode it, generating the package sequence for each robot. Then, we check the sequence to see if the accumulation of weight is over the maximum payload of the robot. If so, 0 is inserted into the sequence. As 0 is the label of the warehouse, this will force the robot to go back to the warehouse at a proper time. In this way, if the total weight of packages for a robot is over its maximum payload, the robot will divide the packages into several batches. From A^{distance}, we can get the shortest distance for a robot to finish the task following this sequence. Then, the time spent by each robot to complete its assignment can be obtained, and finally, we get the value of the fitness function.

In order to guide robots, we then calculate the waypoint sequences for the robots and the actions taken at each waypoint, listed in lines 29–48. This is achieved by calculating the shortest sequence of waypoints that connect each pair of nodes in the task sequence according to A^{via}.

Input: $A^{\text{map}}, A^{\text{robot}}, A^{\text{package}}$
Output: $A_i^{\text{out}}, i = 1, 2, \ldots, N$
(1) Create A^{distance} with $A^{\text{distance}} \leftarrow A^{\text{map}}$
(2) Create A^{via} with $a_{i,j}^{\text{via}} \leftarrow j \forall i = 0, \ldots, L, j = 0, \ldots, L$
(3) **For** $v = 0 \longrightarrow L$, **do**
(4) **For** $a = 0 \longrightarrow L$, **do**
(5) **For** $b = 0 \longrightarrow L$, **do**
(6) **If** $a_{a,b}^{\text{distance}} > a_{a,v}^{\text{distance}} + a_{v,b}^{\text{distance}}$, **then**
(7) $a_{a,b}^{\text{distance}} \leftarrow a_{a,v}^{\text{distance}} + a_{v,b}^{\text{distance}}$
(8) $a_{a,b}^{\text{via}} \leftarrow a_{a,v}^{\text{via}}$
(9) **End if**
(10) **End for**
(11) **End for**
(12) **End for**
(13) Create swarm: $p_i \leftarrow \text{RAND}(0, N), i = 1, 2, \ldots, n_{\text{swarm}}$, set $v_i \leftarrow 0, i = 1, 2, \ldots, n_{\text{swarm}}$
(14) $p_i^{\text{best}} \leftarrow p_i$, $i = 1, 2, \ldots, n_{\text{swarm}}, g^{\text{best}} \leftarrow p_1$
(15) **For** $i = 1 \longrightarrow n_{\text{iter}}$, **do**
(16) **For** $j = 1 \longrightarrow n_{\text{swarm}}$, **do**
(17) $v_j \leftarrow c_0 \times v_j + c_1 \times \text{RAND}(0, 1) \times (p_{\text{best}} - p_j)$
(18) $+ c_2 \times \text{RAND}(0, 1) \times (g_{\text{best}} - p_j)$
(19) $p_{\text{temp}} \leftarrow p_j + v_j$
(20) Constrain each element of P_{temp} within $(0, N)$,
(21) $P_j \leftarrow P_{\text{temp}}$
(22) **If** $\text{FITNESS}(P_j^{\text{best}}) > \text{FITNESS}(P_j)$, **then**
(23) $P_j^{\text{best}} \leftarrow P_j$
(24) **End if**
(25) $g^{\text{best}} \leftarrow \text{argmin}_{P^{\text{best}}}(\text{FITNESS}(P^{\text{best}}))$
(26) **End for**
(27) **End for**
(28) $T_i \leftarrow \text{DECODE}(g^{\text{best}}), i = 1, 2, \ldots, N$
(29) **For** $i = 1 \longrightarrow N$, **do**
(30) $t_{\text{start}} \leftarrow t_1, t_1 \in T_i$
(31) $S^{\text{path}} \leftarrow [t_{\text{start}}]$
(32) **If** t start $= 0$, **then**
(33) $S^{\text{action}} \leftarrow [\text{'}pp\text{'}]$
(34) **Else**
(35) $S^{\text{action}} \leftarrow [\text{'}dp\ \text{'}]$
(36) **End if**
(37) **For** $j = 2 \longrightarrow \text{LENGTH}(T_i)$, **do**
(38) $t_{\text{end}} \leftarrow t_j$
(39) $S_{\text{temp}}^{\text{path}} \leftarrow \text{PATH}(t_{\text{start}}, t_{\text{end}}, A^{\text{via}})$
(40) Delete the first element from S_{temp}
(41) $S^{\text{path}} \leftarrow [S^{\text{path}}, S_{\text{temp}}^{\text{path}}]$
(42) $S_{\text{temp}}^{\text{action}} \leftarrow [\text{'}-\text{'}, \ldots, \text{'}-\text{'}]$, the number of '-' is the same as $\text{LENGTH}(S_{\text{temp}})$
(43) $t_{\text{start}} \leftarrow t_{\text{end}}$
(44) $S^{\text{path}} \leftarrow [S^{\text{path}}, t_{\text{start}}]$
(45) **If** $t_{\text{start}} = 0$, **then**
(46) $S^{\text{action}} \leftarrow [S^{\text{action}}, \text{'}pp\text{'}]$
(47) **Else**
(48) $S^{\text{action}} \leftarrow [S^{\text{action}}, \text{'}dp\ \text{'}]$
(49) **End if**
(50) **End for**
(51) **End for**
(52) $A^{\text{out}} \leftarrow [S^{\text{path}}; S^{\text{action}}]$

Algorithm 1: Algorithm to assign tasks and plan path.

In this algorithm, the $\text{RAND}(a, b)$ function returns a random real number between a and b. The function $\text{DECODE}(P)$ decodes a particle P and returns N sequences, and each of them represents a task sequence for a robot. The function $\text{LENGTH}(P)$ returns the length of vector P. The function $\text{PATH}(v_{\text{start}}, v_{\text{end}}, A^{\text{via}})$ provides a sequence of waypoints, based on Floyd's algorithm. The function $\text{FITNESS}(P)$ offers the fitness value of the particle P, achieved with Algorithm 2. In this algorithm, we first decode the particle to obtain a sequence of tasks, i.e., the

Input: A^{distance}, p, A^{robot}, A^{package}
Output: v_{fitness}
(1) $T_i \leftarrow \text{DECODE}(P), i = 1, 2, \ldots, N$
(2) **For** $i = 1 \longrightarrow N$, **do**
(3) $w_{\text{total}} = w_1$, get w_1 from A^{weight}, $d^{\text{total}} = 0$
(4) **For** $j = 2 \longrightarrow \text{LENGTH}(T_i)$, **do**
(5) $w_{\text{total}} \leftarrow w_{\text{total}} + w_j$
(6) **If** $w_{\text{total}} > w_i^{\text{payload}}$, **then**
(7) Get w_i^{payload} from A^{robot}
(8) Insert 0 after the j^{th} element in T_i
(9) $d^{\text{total}} \leftarrow d^{\text{total}} + A^{\text{distance}}_{t_{(j-1,0)}}$
(10) $w_{\text{total}} \leftarrow 0$
(11) **Else**
(12) $d^{\text{total}} \leftarrow d^{\text{total}} + A^{\text{distance}}_{t_{(j-1,j)}}$
(13) **End if**
(14) **End for**
(15) $\text{time}_i \leftarrow d^{\text{total}} \div \text{speed}_i$
(16) Get speed_i from A^{package}
(17) **End for**
(18) $v_{\text{fitness}} \leftarrow \max(\text{time}_1, \text{time}_2, \ldots, \text{time}_N)$.

Algorithm 2: Fitness function.

destinations of packages. We then insert 0, which indicates the warehouse, into proper positions in the sequence so that the total weight a robot is carrying will not be heavier than its maximum payload. Finally, we calculate the time spent by each robot to accomplish the task and choose the solution with the maximum fitness score as the return value.

5. Coordinate Controller for Distributed Logistic with Multiple UGVs

Based on Algorithm 2, we have developed a controller that can be distributed on a set of UGVs. These controllers can communicate with each other, achieve collective decision-making, and finally drive each robot to deliver goods to corresponding targets following an optimized path.

To further speed up the process of obtaining the work plan, distributed optimization is adopted. With this scheme, the computational load is divided into different UGVs in the system. Then, a semiphysical simulation platform is developed, with a workstation working as the working environment that executes robot models. We randomly generate various initial conditions, i.e., random maps, features of robots, and weight of packages, to compare the performance of the coordinate controller. Finally, we install these coordinate controllers to a set of UGVs and carry out experiments in a simulated environment. As the simulated environment is very similar to the actual working scenario, the experiment can demonstrate the effectiveness of the controller.

Even through each controller in the swarm has a coordinator, at any time, only one of them acts as a leader. The leader keeps sending heartbeat packages, and once it fails, the whole swarm elects a new leader. To ensure only one robot is elected as the leader, we need to map some kind of unique ID into a metric that can be sorted or compared. In our work, we use the MAC address as a unique ID for each robot, and we map a MAC address into a number, which is contained in the heartbeat package of the leader. In case that a robot receives no heartbeat package for a period, it broadcasts a heartbeat package to its peers. If a robot receives a heartbeat package with a smaller ID, it is forced into silence, acting as a follower. Finally, only the robot with the smallest ID could emit and becomes the leader.

The leader robot is in charge of controlling the optimization process of the whole swarm, so that all robots can agree to the same work plan. There are several triggers that can start the optimization process for the swarm. The trigger can be any situation that changes conditions of the last optimization, such as failure of a robot, change of destination, or arrival of new packages. Once conditions change, a new plan should be obtained as fast as possible. Thus, the coordinator sends a stop message a period after sending a start signal. This period should be short, and in our case, we set it as one minute, which can be adjusted according to application requirements.

In order to avoid contradiction, all robots in the swarm should agree to the same work plan. The leader coordinator determines the current best particle as a work plan and broadcasts it to the whole swarm.

A PSO engine that runs on each robot is shown in Figure 2. During iterations, each time a robot finds a better solution, it broadcasts the solution to the swarm. Once receiving such message, a robot will compare the solution with its current global best solution. If it is a better solution, then the current global best solution is replaced. It can be proved that ignoring package loss, each PSO engine always agrees with the best particle of the whole swarm. The problem can be abstracted into the following mathematical problem.

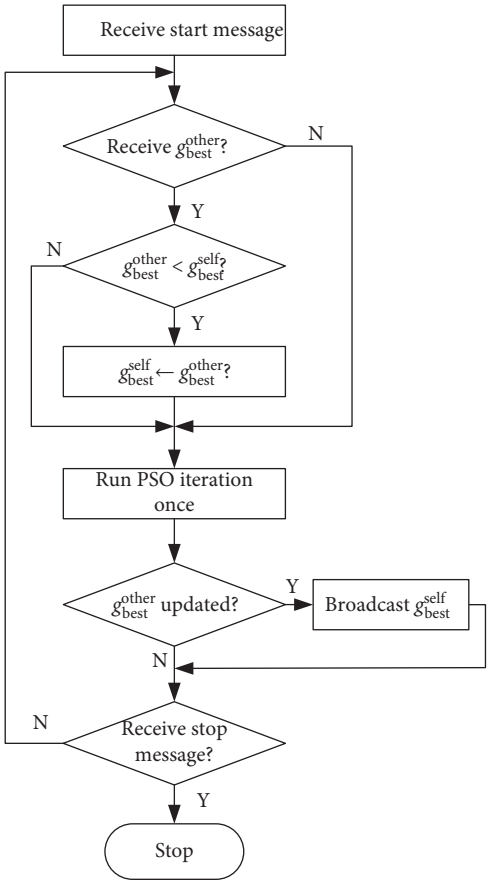

Figure 2: Flow chart of PSO engine.

There are N arrays recorded as (P_i, Pb_i), $i = 1, \ldots, N$. P_i is a parcel of the i-th robot, which is generated periodically. Pb_i is the best solution of the i-th robot. For each array, P_i randomly changed, and it is not necessary for each P_i of all robots to change synchronously. The change of Pb_i obeys the following rules:

(1) Initially, $Pb_i(0) \leftarrow \infty$

(2) Pb_i is updated as $Pb_i = P_i^{new}$ if $P_i^{new} \leq Pb_i$

(3) When any Pb_i in a robot is updated, $Pb_j \leftarrow Pb_i$, if $Pb_j > Pb_i$, $\forall j \in \{1, \ldots, N\}$, $j \neq i$.

It can be ensured that the system can converge to $Pb_{best} = Pb_i, i = 1, \ldots, N$. Pb_{best} is the smallest number that has been generated so far for the swarm. Strict proof is given as follows.

Assume array I generates a P_i^{new} and $P_i^{new} \leq Pb_i$; according to rule (2) above, P_i is updated only when $P_i^{new} \leq Pb_i$. According to rule (3), if $Pb_j > Pb_i$, Pb_i is not updated, and thus we have that the current Pb_i is smaller than any P_i ever generated. Assume at T, $Pb_i(T)$ is updated. Meanwhile, $Pb_j(T), \forall j \in \{1, \ldots, N\} \backslash i$ will also try to update according to rule (3). There are two situations:

(1) If $Pb_j(T) \geq Pb_i(T)$, then $Pb_j(T) \leftarrow Pb_i(T)$ so $Pb_j(T) = Pb_i(T)$.

(2) If $Pb_j(T) < Pb_i(T)$, according to rule (1), this implies that $Pb_j(T) \neq \infty$. So, $Pb_j(T)$ has at least been updated once. Assume that the last time Pb_j is updated at $T - 1$. Thus, the value of Pb_j is marked as $Pb_j(T-1)$, and consequently, the value of Pb_i is $Pb_i(T-1)$. If $Pb_i(T-1) > Pb_j(T-1)$, then $Pb_i(T-1)$ should be updated according to rule (3), and we have $Pb_i(T-1) = Pb_j(T-1) = Pb_j(T) < Pb_i(T)$. Then, we have $Pb_i(T-1) < Pb_i(T)$. According to rule (2), Pb_i should not be updated at T.

Therefore, the second situation mentioned above is false and we can directly get the solution that $\forall i, Pb_i$ is the same value. Thus, Pb_i is the smallest value ever generated.

Mapping the problem mentioned above to our distributed PSO algorithm, we have that all PSO engines share the same g_{best}, and thus, it is similar to the PSO algorithm running on the same computer. Besides, according to the problem mentioned above, synchronization is not necessary. Thus, the controller can run on a set of heterogeneous robots.

6. Simulation Study

6.1. Semiphysical Simulation. Based on the abovementioned ideas, we developed the controller using Raspberry Pi 3b+. This controller is called the distributed logistic controller (DLC). To help develop the controller and test its performance, we also developed a semiphysical simulation platform. Then, we randomly generated multiple maps with packages with different weights, and simulation experiments were carried out to compare the performance of the classic centralized PSO controller and the DLC.

The design of the semiphysical simulation platform is shown in Figure 3. It is developed with Qt Creator and is running on a Dell Precision 3630 workstation. It imitates the environment and dynamics models of UGVs. Multiple Raspberry Pi boards connect the simulation platform via Ethernet using a switch. Each Raspberry Pi runs a PSO optimizer engine to find optimized solutions for all robots, including task assignment and path planning. The DLC is also implemented with a Raspberry Pi to verify the system performance.

The simulation platform consists of the following parts:

(1) UI interface, displaying the status of the robots and allowing people to control the simulation process

(2) Robot models, equipped with the low layer of a PID controller and dynamic model of UGVs, so that the platform can simulate the performance of the UGVs

(3) Simulation engine, basic structure of the simulation platform

(4) Virtual environment, loaded map that can interact with the robot models

(5) Virtual-physical interface, used to connect the robot controller into the simulation platform.

Besides debugging the program during the development of the controller, we used the platform to test the performance of the controller. The simulation platform is given in Figure 3. The reason why we developed a distributed

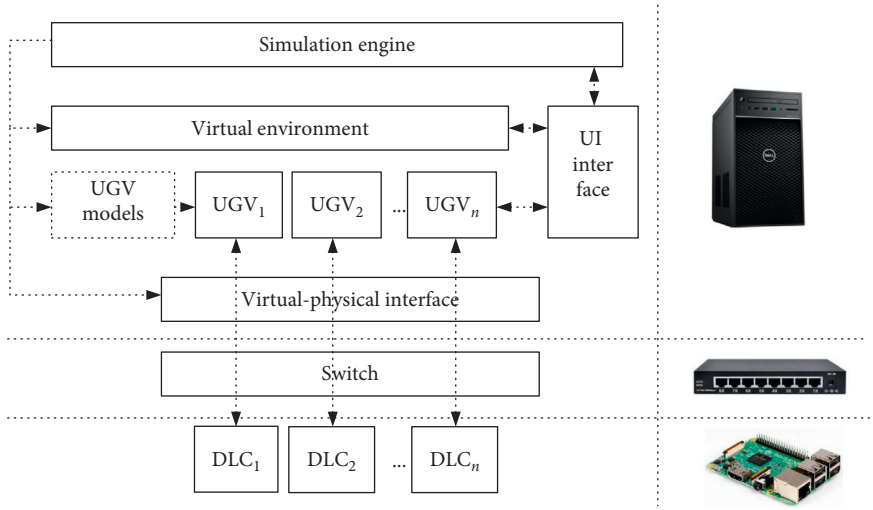

Figure 3: Simulation platform.

controller is to deal with the situation where a high performance workstation is unavailable. The simulation process is shown in Figure 4.

Various experiments were carried out on the half-physical simulation platform to test the performance of the controller. For verifying the results, the adaptive cat swarm optimization (ACSO) in [55] and the classic centralized logistic controller are used to compare with the proposed distributed controller. As the optimization algorithm has a feature of randomness, we generate multiple maps randomly and repeat simulation of each map multiple times. The maps were generated with a different number of packages, i.e., 50 packages, 100 packages, 150 packages, and 200 packages, each with 5 maps. Meanwhile, we repeat a simulation 10 times to get a statistical conclusion. To ensure the existence of feasible solutions, all maps are connected graphs. Figure 4 shows one map structure.

For each map, three kinds of controllers are tested. The first controller is a classic centralized PSO controller, the second one is the ACSO controller in [55], and the last one is our DLC controller. For each simulation, we use a robot swarm consisting of four UGVs to deliver packages. The weight of each package is randomly generated along with each map. The weights are within the range of $(0, w_{\max})$, where $w_{\max}$ is the maximum payload capability among all the robots. These hypotheses ensure the existence of a feasible solution. The UGVs are heterogeneous, and their features are shown in Table 1.

With the conditions mentioned above, we carried out 600 experiments in total. As real controllers are connected into the emulation platform, the simulation results represent the actual performance and the working condition of the controller. For all the experiments, the DLC controllers always provided a feasible solution, with no failure occurring. This indicates that the DLC controller can work reliably and can be directly used on real robots. The simulation results are shown in Figure 5. It indicates that DLC provides better solutions than the classic PSO controller. As shown in the figures, the experimental results of the classic PSO controller are painted in blue, the experimental results of the ACSO controller are painted in green, and the performance with the DLC is painted in red. It is obvious that the mean delivery time using the DLC is the shortest among all three controllers. Furthermore, in most cases, the delivery time variation of experiments is smaller with DLCs than that with classic PSO controllers or with the ACSO controller, which improves the system performance.

The time complexity of the proposed DLC can be expressed as $O(N_{\text{iter}} \times N_{\text{swarm}} \times N_{\text{robot}} \times N_{\text{package}})$, with N_{iter} being the number of iterations, N_{swarm} being the swarm size, N_{robot} being the number of robots deployed, and N_{package} being the number of packages to be delivered. Besides time complexity, we also care about the convergence performance of the solution provided by the DLC. Facing a specific NP-hard problem, whose analytical optimum solution is hard to be obtained, we cannot guarantee to obtain the global optimum solution. However, we can verify the extent to which it performs better than available methods with a large number of experiments. Furthermore, statistical conclusions can be drawn based on experiment results. We define a metric as $p_{\text{norm}} = ((s^{\text{other}}_{\text{mean}} - s^{\text{DLC}}_{\text{mean}})/s^{\text{DLC}}_{\text{mean}}) \times 100\%$. For the same set of experimental conditions, we can obtain a set of solutions. $s^{\text{other}}_{\text{mean}}$ is the mean value of solutions obtained with either a PSO controller or the ACSO controller, and $s^{\text{DLC}}_{\text{mean}}$ is the mean value of the solution obtained with the DLC. It represents a normalization of the extra mean task makespan taking the mean value of the DLC as a benchmark. It turns out that compared with a PSO controller, the values of p_{norm} for maps with 50, 75, 100, and 150 packages are 4.87 %, 12.51 %, 10.28 %, and 8.95 %, respectively. Compared with the ACSO controller, the values of p_{norm} for maps with 50, 75, 100, and 150 packages are 11.80 %, 16.50 %, 11.63 %, and 10.79 %, respectively. As time spent on every experiment is the same, the results indicated that the DLC has a higher chance to converge faster than the other two methods in our experiments. We further define the metric $s_{\text{norm}} = (s_{\text{std}}/s_{\text{mean}}) \times 100\%$, in which s_{std} is the standard deviation of a set of solutions, and s_{mean} is the mean value of this set of

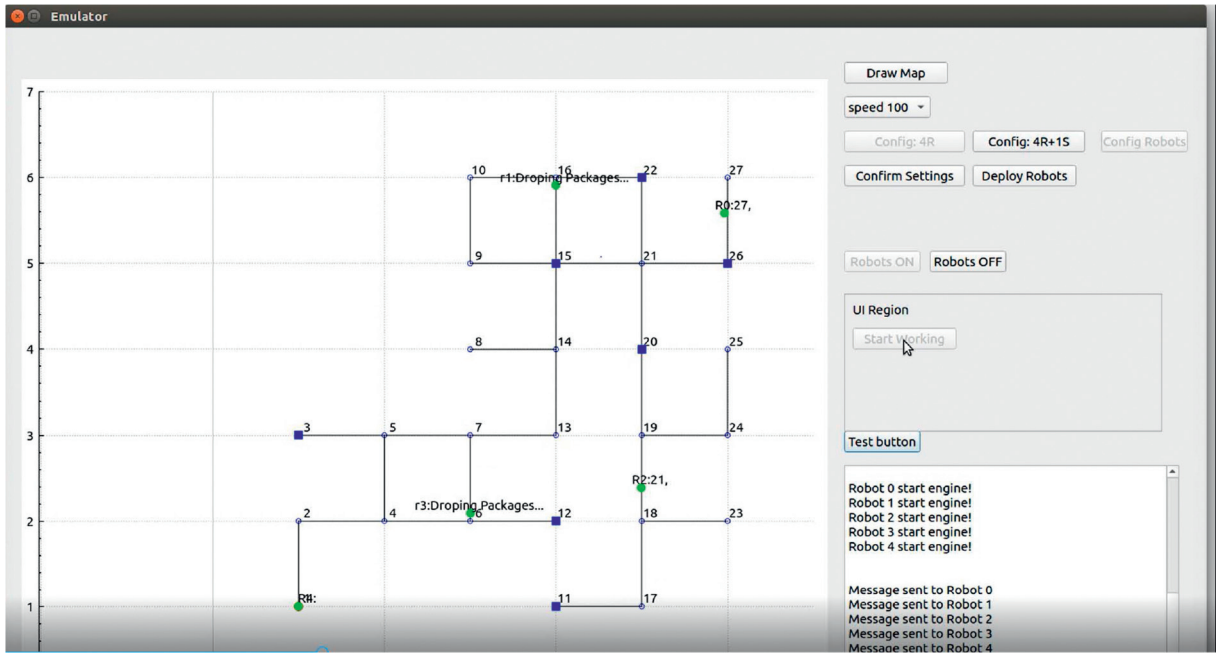

Figure 4: Simulation process. Green dots stand for robots, blue squares mean that parcels have been delivered to destinations, and blue circles are destinations; a text above a green dot indicates the robot ID, the current loaded goods, and the current action.

Table 1: Robot features of a semiphysical simulation.

Robot ID	1	2	3	4
Speed (m/s)	2	1	2.5	1
Payload (kg)	35	20	50	70
Load time (s)	10	10	15	15
Drop time (s)	10	10	15	15

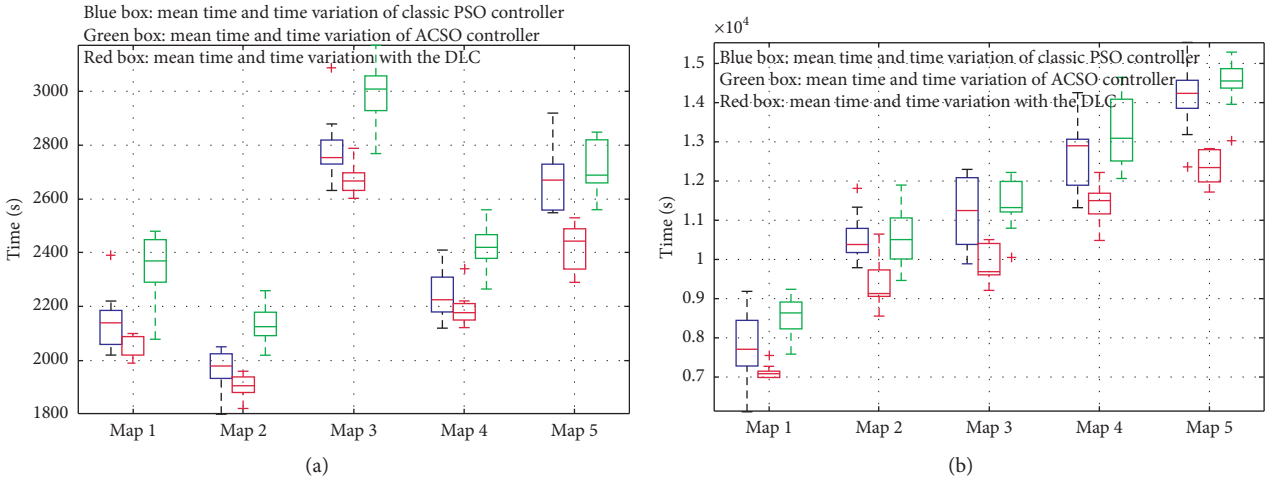

Figure 5: Continued.

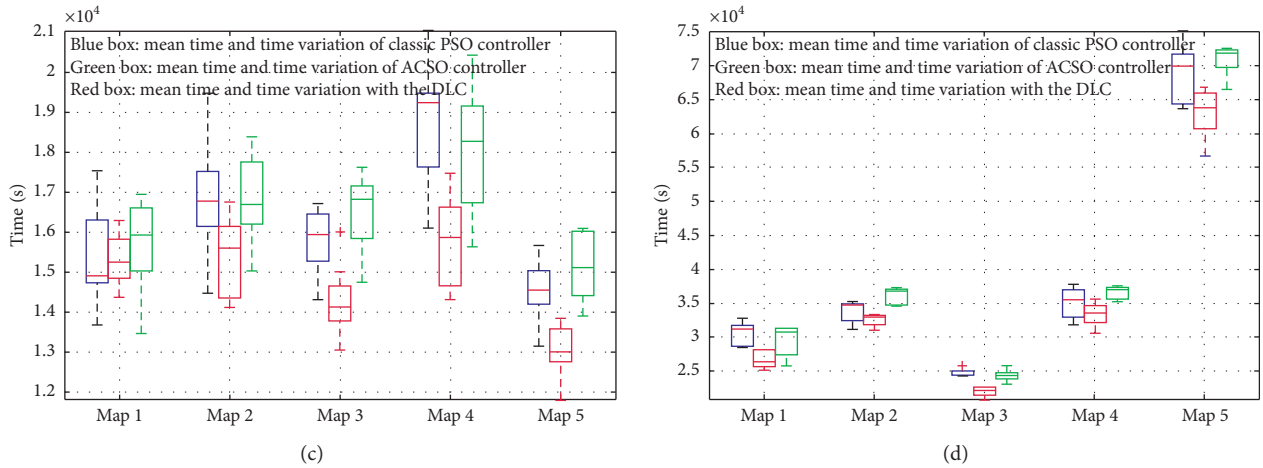

Figure 5: Simulation results: (a) simulation with 50 packages using the classic centralized PSO controller, the ACSO controller, and the DLC in 5 randomly generated maps; (b) simulation with 75 packages using the classic centralized PSO controller, the ACSO controller, and the DLC in 5 randomly generated maps; (c) simulation with 100 packages using the classic centralized PSO controller, the ACSO controller, and the DLC in 5 randomly generated maps; (d) simulation with 150 packages using the classic centralized PSO controller, the ACSO controller, and the DLC in 5 randomly generated maps.

solutions. This metric is a normalization of the standard deviation of a set of solutions. For the DLC, the values of s_{norm} for maps with 50, 75, 100, and 150 packages are 2.57%, 4.34%, 5.66%, and 4.78%, respectively. The values indicate that the DLC can provide stable solutions in spite of changes in maps.

6.2. Experiment with UGVs. Finally, we installed these controllers into UGVs and experiments were carried out. In these experiments, goods are delivered by three different robots. The features of the robots are shown in Table 2. In the table, we report the maximum payload and the speed of each robot. We assume that each robot spends constant time to pickup or drop a package. The time is also shown in the table. The arena consists of 15 houses and 1 warehouse, with the map shown in Figure 6.

We assume a truck will carry all the packages and the robots to the warehouse, and then robots are sent out to deliver each package to its destination. We assign label 0 to the warehouse, and the labels of the houses increase from 1 to 15. The weight of the packages for each house is shown in Table 3.

The algorithm proposed in this paper was used to solve the task assignment and path planning problems for a multirobot system. We assign the tasks for each robot and plan the paths for them to follow, so that the task can be accomplished in the shortest time. In order to achieve this purpose, we first build a graph model according to the map shown in Figure 6. Both the warehouse and the houses are treated as nodes. If a robot can move from one node directly to another node, without passing by a third node, we connect the two nodes with a edge, with the distance between the two nodes assigned as the weight of the edge. The graph model of the environment is shown in Figure 7. The adjacency matrix can then be built based on the graph model.

Table 2: Robot feature.

Robot ID	1	2	3
Speed (m/s)	6	3	7.5
Payload (kg)	70	190	30
Load time (s)	45	60	20
Drop time (s)	30	30	10

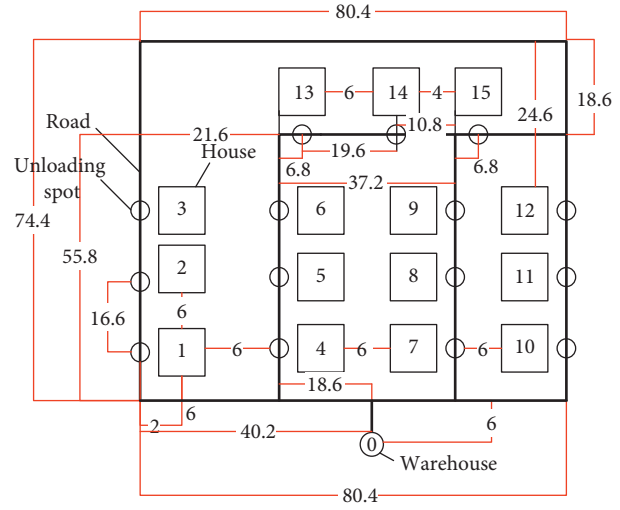

Figure 6: Map of the arena.

For the PSO algorithm, we set the swarm size as 50 and the total iteration time as 200. The minimum time to complete all tasks is about 727 s, and this solution can be obtained within 30 seconds. This makespan value does not vary much, even over a large number of iterations. We also obtain the tasks and paths for each robot, as shown in Tables 4, 5, and 6. In these tables, the waypoint sequences are given for a robot to follow, and its corresponding behavior is

Table 3: Package weight for each house.

House ID	Package weight (kg)
1	5
2	20
3	15
4	50
5	33
6	45
7	70
8	30
9	15
10	5
11	5
12	10
13	5
14	40
15	12

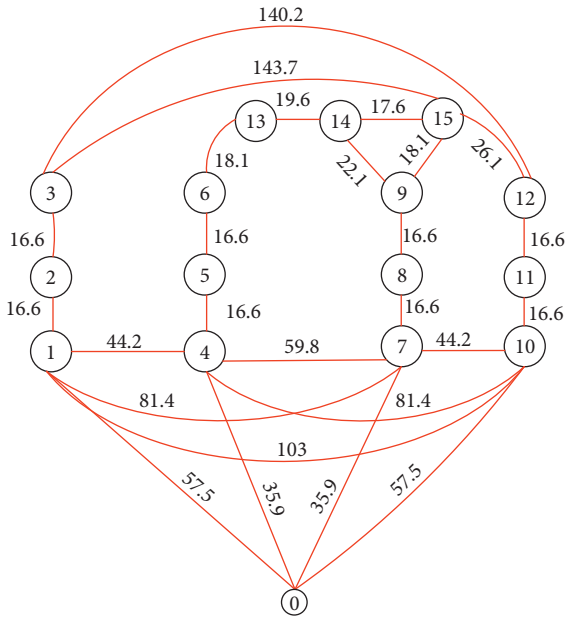

Figure 7: Graph model.

Table 4: Path and tasks for robot 1.

Waypoint	Behavior
0	pp
12	—
7	dp
12	—
0	pp
2	—
8	—
11	dp
8	dp
2	—
0	pp
6	—
5	—
4	—

Table 4: Continued.

Waypoint	Behavior
13	—
16	—
14	dp
16	—
13	—
4	—
5	—
6	—
0	pp
6	dp
0	—

Table 5: Path and tasks for robot 2.

Waypoint	0	6	5	4	18	19	20	11	8	2	0
Behavior	pp	—	dp	dp	dp	dp	dp	—	—	—	—

Table 6: Path and tasks for robot 3.

Waypoint	Behavior
0	pp
6	—
5	—
4	—
13	—
16	-
14	—
17	dp
15	dp
17	—
14	—
16	—
13	—
4	—
5	—
6	—
0	pp
1	dp
2	dp
0	pp
12	—
7	—
3	dp
7	pp
12	—
1	—
9	dp
1	—
0	pp
12	dp
7	—
3	—
18	—
13	dp
10	pp
9	—
1	—
0	—

Figure 8: Photo of the physical simulation scene.

also listed, such as pickup packages (pp), drop packages (dp), or just pass by the waypoint (—). The physical simulation scene is shown in Figure 8.

7. Conclusion

In this paper, we consider the last-mile delivery problem with multiple heterogeneous unmanned ground vehicles (UGVs), which is formulated as an optimization problem aimed at minimizing the maximum makespan. The proposed algorithm combines the Floyd's algorithm and the particle swarm optimization (PSO) algorithm for task assignment and path planning. A proposed distributed logistic controller enables UGVs to achieve excellent solutions in dynamic environments. A distributed logistic controller (DLC) is developed to control multiple UGVs. Semiphysical simulation results with the DLC have the shortest mean delivery time compared with using a classic PSO controller and the ACSO controller. In most cases, the observed experimental delivery time variations are smaller with DLCs than that with classic PSO controllers or with the ACSO controller. Experiments with UGV prototypes were carried out. The results confirm the validity and applicability of the developed approach. Goods are delivered with optimized makespan obtained with the proposed algorithm.

Abbreviations

UGV: Unmanned ground vehicle.
PSO: Particle swarm optimization.
VRP: Vehicle routing problem.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Yuzhan Wu conceptualized the study and developed methodology. Yuzhan Wu, Yuanhao Ding, Susheng Ding, and Meng Li validated the study. Yuzhan Wu wrote the original draft. Yvon Savaria and Meng Li reviewed and edited the article. All authors have read and agreed to the published version of the manuscript.

References

[1] M. Wei, S. Wang, J. Zheng, and D. Chen, "UGV navigation optimization aided by reinforcement learning-based path tracking," *IEEE Access*, vol. 6, pp. 57814–57825, 2018.

[2] L. A. Curiel-Ramirez, R. A. Ramirez-Mendoza, R. Bautista-Montesano et al., "End-to-end automated guided modular vehicle," *Applied Sciences*, vol. 10, no. 12, p. 4400, 2020.

[3] T. Kato and R. Kamoshida, "Multi-Agent simulation environment for logistics warehouse design based on self-contained agents," *Applied Sciences*, vol. 10, no. 21, p. 7552, 2020.

[4] X. Wang, T.-M. Choi, H. Liu, and X. Yue, "Novel ant colony optimization methods for simplifying solution construction in vehicle routing problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3132–3141, 2016.

[5] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[6] A. Rahimi-Vahed, T. G. Crainic, M. Gendreau, and W. Rei, "A path relinking algorithm for a multi-depot periodic vehicle routing problem," *Journal of Heuristics*, vol. 19, no. 3, pp. 497–524, 2013.

[7] V. N. Coelho, A. Grasas, H. Ramalhinho, I. M. Coelho, M. J. F. Souza, and R. C. Cruz, "An ILS-based algorithm to solve a large-scale real heterogeneous fleet VRP with multi-trips and docking constraints," *European Journal of Operational Research*, vol. 250, no. 2, pp. 367–376, 2016.

[8] S. Somashekhara, A. Setty, S. Sridharmurthy, P. Adiga, U. Mahabaleshwar, and G. Lorenzini, "Makespan reduction using dynamic job sequencing combined with buffer optimization applying genetic algorithm in a manufacturing system," *Mathematical Modelling of Engineering Problems*, vol. 6, no. 1, pp. 29–37, 2019.

[9] G. Goyal and S. Vadhera, "Solution of combined economic emission dispatch with demand side management using metaheuristic algorithms," *Journal Européen des Systèmes Automatisés*, vol. 52, no. 2, pp. 143–148, 2019.

[10] L. Bai and C. Du, "Design and simulation of a collision-free path planning algorithm for mobile robots based on improved ant colony optimization," *Ingénierie des Systèmes d Information*, vol. 24, no. 3, p. 331, 2019.

[11] A. K. Beheshti and S. R. Hejazi, "A novel hybrid column generation-metaheuristic approach for the vehicle routing problem with general soft time window," *Information Sciences*, vol. 316, pp. 598–615, 2015.

[12] F. Yan and Y. Wang, "Modeling and solving the vehicle routing problem with multiple fuzzy time windows," in *Proceedings of the International Conference on Management Science and Engineering Management*, pp. 847–857, Kanazawa, Japan, July 2017.

[13] B. Eksioglu, A. V. Vural, and A. Reisman, "The vehicle routing problem: a taxonomic review," *Computers & Industrial Engineering*, vol. 57, no. 4, pp. 1472–1483, 2009.

[14] H. C. Lau, T. Chan, W. Tsui, and W. Pang, "Application of genetic algorithms to solve the multidepot vehicle routing problem," *IEEE Transactions on Automation Science and Engineering*, vol. 7, pp. 383–392, 2009.

[15] Q. Meng, D.-H. Lee, and R. L. Cheu, "Multiobjective vehicle routing and scheduling problem with time window constraints in hazardous material transportation," *Journal of Transportation Engineering*, vol. 131, no. 9, pp. 699–707, 2005.

[16] G. Kim, Y.-S. Ong, C. K. Heng, P. S. Tan, and N. A. Zhang, "City vehicle routing problem (city VRP): a review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1654–1666, 2015.

[17] Y. Wang, Y. He, L. He, and L. Xing, "Bio-inspired algorithms applied in multi-objective vehicle routing problem: frameworks and applications," in *Proceedings of the 10th International Conference Bio-Inspired Computing-Theories and Applications*, Hefei, China, September 2015.

[18] X. Zhu, R. Yan, Z. Huang, W. Wei, J. Yang, and S. Kudratova, "Logistic optimization for multi depots loading capacitated electric vehicle routing problem from low carbon perspective," *IEEE Access*, vol. 8, pp. 31934–31947, 2020.

[19] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1135–1145, 2015.

[20] E. W. Dijkstra and others, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[21] Z. Zhan, L. Jin, Z. Cao, S. Jiao, and N. Bai, "Research on mine emergency rescue command system based on chambers," *Procedia Engineering*, vol. 45, pp. 710–715, 2012.

[22] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[23] X. Wang, *Study on the Algorithm of Minimum Time Consumption in Urban Transportation Network*, Master's Thesis, Hebei University, Baoding, China, 2018.

[24] A. Bozyigit, G. Alankus, and E. Nasiboglu, "Public transport route planning: modified Dijkstra's algorithm," in *Proceedings of the 2017 International Conference on Computer Science and Engineering (UBMK)*, Antalya, Turkey, October 2017.

[25] K. Wei, Y. Gao, W. Zhang, and S. Lin, "A modified Dijkstra's algorithm for Solving the problem of finding the maximum load path," in *Proceedings of the 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, Kahului, HI, USA, March 2019.

[26] Y. Jin and G. Xiaoxue, "Optimal route planning of parking lot based on Dijkstra algorithm," in *Proceedings of the International Conference on Robots & Intelligent System*, Huaian, China, October 2017.

[27] H. Luo, J. Yang, and X. Nan, "Path and transport mode selection in multimodal transportation with time window," in *Proceedings of the 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 162–166, Chongqing, China, October 2018.

[28] Z. Nie and H. Zhao, "Research on robot path planning based on Dijkstra and Ant colony optimization," in *Proceedings of the 2019 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, pp. 222–226, Shanghai, China, 2019.

[29] M. Zhou, X. Qu, and W. Qi, "Improving efficiency at highway T-junctions with connected and automated vehicles," *Transportmetrica A: Transport Science*, vol. 17, no. 1, pp. 107–123, 2021.

[30] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3310–3317, San Diego, CA, USA, May 1994.

[31] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: the field D∗ algorithm," *Journal of Field Robotics*, vol. 23, no. 2, pp. 79–101, 2006.

[32] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[33] T. Chen, G. Zhang, X. Hu, and J. Xiao, "Unmanned aerial vehicle route planning method based on a Star algorithm," in *Proceedings of the 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1510–1514, Wuhan, China, May 2018.

[34] X. Bai, P. Wang, Z. Wang, and L. Zhang, "3D multi-UAV collaboration based on the hybrid algorithm of Artificial Bee Colony and A∗," in *Proceedings of the 2019 Chinese Control Conference (CCC)*, pp. 3982–3987, Guangzhou, China, 2019.

[35] L. Wang and S. Pang, *Chemical Plume Tracing Using an AUV Based on POMDP Source Mapping and A-Star Path Planning*, , pp. 1–7, OCEANS 2019 MTS/IEEE SEATTLE, 2019.

[36] M. S. Montemerlo, J. Becker, S. Bhat et al., "The Stanford entry in the urban challenge," *Journal of Field Robotics*, 2008.

[37] O. Pink, C. Frese, and C. Stiller, "Team AnnieWAY's autonomous system for the 2007 DARPA urban challenge," *Journal of Field Robotics*, vol. 25, pp. 615–639, 2010.

[38] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, pp. 939–960, 2010.

[39] J. Wu, B. Kulcsár, S. Ahn, and X. Qu, "Emergency vehicle lane pre-clearing: from microscopic cooperation to routing decision making," *Transportation Research Part B: Methodological*, vol. 141, pp. 223–239, 2020.

[40] J. Wu, S. Ahn, Y. Zhou, P. Liu, and X. Qu, "The cooperative sorting strategy for connected and automated vehicle platoons," *Transportation Research Part C: Emerging Technologies*, vol. 123, Article ID 102986, 2021.

[41] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H.-M. Tai, "Autonomous local path planning for a mobile robot using a genetic algorithm," *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 2, pp. 1338–1345, 2004.

[42] H. J. Bremermann, *The Evolution of Intelligence: The Nervous System as a Model of its Environment*, University of Washington, Department of Mathematics, Seattle, WA, USA, 1958.

[43] T. Shibata and T. Fukuda, "Intelligent motion planning by genetic algorithm with fuzzy critic," in *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pp. 565–570, Chicago, IL, USA, August 1993.

[44] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 18–28, 1997.

[45] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International conference on Neural Networks*, pp. 1942–1948, Perth, Australia, 1995.

[46] X.-l. Tang, L.-m. Li, and B.-j. Jiang, "Mobile robot SLAM method based on multi-agent particle swarm optimized particle filter," *The Journal of China Universities of Posts and Telecommunications*, vol. 21, no. 6, pp. 78–86, 2014.

[47] E. Masehian and D. Sedighizadeh, "Multi-objective PSO- and NPSO-based algorithms for robot path planning," *Advances in Electrical and Computer Engineering*, vol. 10, no. 4, pp. 69–76, 2010.

[48] Y. Wang, P. Bai, X. Liang, W. Wang, J. Zhang, and Q. Fu, "Reconnaissance mission conducted by UAV swarms based on distributed PSO path planning algorithms," *IEEE Access*, vol. 7, pp. 105086–105099, 2019.

[49] Z. Yan, J. Li, Y. Wu, and G. Zhang, "A real-time path planning algorithm for AUV in unknown underwater environment based on combining PSO and waypoint guidance," *Sensors*, vol. 19, p. 20, 2019.

[50] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: an 5 autocatalytic optimizing process," *Technical Report 91-016*, pp. 1–27, 1991.

[51] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[52] Y. Zhang, H. L. Wang, X. Li, Y. H. Zhang, and H. Wang, "Parallel ant system based on OpenMP," *Advanced Materials Research*, vol. 765-767, pp. 658–661, 2013.

[53] T. Stützle and H. H. Hoos, "MAX–MIN- ant system," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[54] B. Bullnheimer, R. F. Hartl, and C. Strauss, "A new rank based version of the Ant system-a computational study," *Central European Journal of Operations Research*, vol. 7, no. 1, pp. 25–38, 1999.

[55] X. F. Ji, J. S. Pan, S. C. Chu, P. Hu, Q. W. Chai, and P. Zhang, "Adaptive Cat swarm optimization algorithm and its applications in vehicle routing problems," *Mathematical Problems in Engineering*, vol. 2020, Article ID 1291526, , 2020.